# Notes on application of the Normalized Gradient Descent approach to Deep Learning

Krzysztof Kolasiński*

June 3, 2018

**Abstract**

This notes tackle the problem of separating gradient amplitude i.e. step size from the gradient direction. In this notes I describe things I have tried and what was working for me and things which were not.

## 1 Gradient descent

We want to solve following optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{\theta}, \boldsymbol{X}\right),$$

where $\boldsymbol{\theta}$ are the model parameters e.g. weight matrices in Neural Network and $\boldsymbol{X}$ is our dataset, from which we compute the loss function $\mathcal{L}$. A standard way in Machine Learning is to solve this problem with gradient descent (usually a stochastic variant)

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \lambda^{(k)} \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}^{(k)}, \boldsymbol{X}^{(k)}\right)}{\partial \boldsymbol{\theta}} \equiv \boldsymbol{\theta}^{(k)} - \lambda^{(k)} \mathbf{G}^{(k)}$$

where $\boldsymbol{\theta}^{(k)}$ are the model parameters at k-th iteration, $\boldsymbol{X}^{(k)}$ is the batch of data (e.g. images and the labels) at k-th step, $\lambda^{(k)}$ gradient descent step size i.e. learning rate, which in that case may depend on the iteration.

**Potential problem with standard formulation of the Gradient Descent:**

- **Too small gradients** - if gradients amplitude is too small, the effective learning rate is also very small. This is ok when we are close to the minimum, but in some cases it may result with so-called vanishing gradients problem and learning may significantly slow down or even stop.

*email: krzysztof.kolasinski@fornax.ai

- **Too large gradients** - this is an opposite case when gradients may explode.

Those two points show that learning step is somehow double-parametrized since the effective step in $\theta$ space will depend on: a) the value of $\lambda_k$ and b) gradient norm. Gradient normalization helps to reduce this potentially problematic double-parametrization of the effective step size.

In the second order methods the learning rate is replaced by the inverse of the Hessian $\mathbf{H}\left(\mathcal{L}^{(k)}\right)$ matrix like in Newton's method.

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mathbf{H}\left(\mathcal{L}^{(k)}\right)^{-1}\mathbf{G}^{(k)},$$

or its approximations.

In 1D Hessian matrix is just a scalar second derivative of the loss function. Let us consider simple exponential loss function like $\mathcal{L}(x) = e^{-\gamma x}$. The gradient descent for such problem is following

$$x^{(k+1)} = x^{(k)} + \lambda\gamma e^{-\gamma x^{(k)}},$$

hence when $x \ll 0$ the step size will be extremely large and when $x \gg 0$ the step size will be almost zero. However when using Newton's method we get following update:

$$x^{(k+1)} = x^{(k)} - \lambda\frac{-\gamma e^{-\gamma x^{(k)}}}{\gamma^2 e^{-\gamma x^{(k)}}} = x^{(k)} + \frac{\lambda}{\gamma},$$

step size does not depends on the current state of $x^{(k)}$. So the multiplication by the inverse of Hessian has normalizing property in that setup.

## 2 Normalized Gradient Descent

Normalized Gradient Descent (NGD) is a natural modification of the standard method in the optimization problems, namely in every step we compute

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \lambda^{(k)}\frac{\mathbf{G}^{(k)}}{\left\|\mathbf{G}^{(k)}\right\|},$$

where $\left\|\mathbf{G}^{(k)}\right\| > 0$ represents some general normalization function. In particular one can try to use:

- $L_\infty$ norm, which normalizes the gradients by

  $$\left\|\mathbf{G}^{(k)}\right\|_\infty = \max_v \left|G_v^{(k)}\right|,$$

  where $G_v^{(k)}$ denotes the $v$-th element of the $\mathbf{G}^{(k)}$ vector. Intuitively, this normalization guarantees that the maximum change of the parameters is equal to the learning rate i.e. $\max_v \left|\theta_v^{(k+1)} - \theta_v^{(k)}\right| = \lambda^{(k)}$.

- $L_2$ norm, which normalizes the gradients by the standard euclidean norm

$$\left\|\mathbf{G}^{(k)}\right\|_2 = \sqrt{\sum_v \left[G_v^{(k)}\right]^2},$$

This normalization makes that the speed of change of the parameters is always proportional to $\lambda^{(k)}$. This normalization is widely used in the literature [CYTOWANIA]. However, In the context of deep neural network it means that the layers with bigger weight matrices will get updated with smaller amplitudes then smaller layers e.g. bias vector. For example let us consider layer with scalar weight $\omega$, hence the gradient to this layer will be also a scalar $g_\omega$, the descent update is

$$\omega^{(k+1)} = \omega^{(k)} - \lambda^{(k)} \text{sign}\left(g_\omega\right),$$

Now, let us consider N dimensional gradient vector of form $\mathbf{g} = [g, \ldots, g]$, which $L_2$ norm $\|\mathbf{g}\|_2 = \sqrt{N}g$, hence the gradient descent step is

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\lambda^{(k)}}{\sqrt{N}} \frac{\mathbf{g}}{g},$$

the effective step for this case is then $\sqrt{N}$ times smaller. This problem may be partially solved by redefining norm as $\sqrt{\frac{1}{N}\sum_v \left(G_v^{(k)}\right)^2}$.

- Standard deviation normalization, defined as

$$\left\|\mathbf{G}^{(k)}\right\|_{\text{std}} = \sqrt{\frac{1}{N}\sum_v \left(G_v^{(k)} - \bar{G}^{(k)}\right)^2},$$

where $\bar{G}^{(k)} = \frac{1}{N}\sum_v G_v^{(k)}$. Surprisingly, this normalization seems to work in some problems better than $L_\infty$.

**Problems with NGD**

- One has take care about the value of the $\lambda^{(k)}$, especially in order to obtain the convergence the learning rate must be zero (or very small) at the end of the training.
- I found $L_2$ less efficient than others, probably because of the mentioned problem.

# 3 An online iterative steepest gradient descent

Steepest gradient descent looks for a solution of the following problem: at each step of the gradient descent find such value of the learning rate which minimizes the loss function at most, i.e.

$$\lambda^{(k)} = \underset{\lambda}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{\theta}^{(k)} - \lambda \frac{\mathbf{G}^{(k)}}{\left\|\mathbf{G}^{(k)}\right\|}, \boldsymbol{X}^{(k)}\right), \tag{1}$$

For the brevity we define a normalized gradient as

$$\widehat{\mathbf{G}}^{(k)} = \frac{\mathbf{G}^{(k)}}{\left\|\mathbf{G}^{(k)}\right\|}.$$

The exact solution of the Eq. (1) at each iteration is impossible, since in most cases there is not analytic solution for this problem. We are going to use the gradient descent to solve the Eq. (1). We are going to make a large assumption that the optimal value of the $\lambda^{(k)}$ does not vary significantly between batches. Let us note that the expression $\boldsymbol{\theta}^{(k)} - \lambda\widehat{\mathbf{G}}^{(k)}$ represents state of the parameters in the next step, hence for the batch at k+1 step, the loss function is given by

$$\lambda^{(k)} = \underset{\lambda}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{\theta}^{(k)} - \lambda\widehat{\mathbf{G}}^{(k)}, \boldsymbol{X}^{(k+1)}\right) \equiv \underset{\lambda}{\operatorname{argmin}} \mathcal{L}\left(\boldsymbol{\theta}^{(k+1)}(\lambda), \boldsymbol{X}^{(k+1)}\right),$$

The derivative of $\mathcal{L}\left(\boldsymbol{\theta}^{(k+1)}(\lambda), \boldsymbol{X}^{(k+1)}\right)$ w.r.t $\lambda$ is then

$$\begin{aligned}\frac{\partial \mathcal{L}\left(\boldsymbol{\theta}^{(k+1)}(\lambda), \boldsymbol{X}^{(k+1)}\right)}{\partial \lambda} &= \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}^{(k+1)}(\lambda), \boldsymbol{X}^{(k+1)}\right)^T}{\partial \boldsymbol{\theta}} \frac{\partial\left(\boldsymbol{\theta}^{(k)} - \lambda\widehat{\mathbf{G}}^{(k)}\right)}{\partial \lambda} \\ &= -\frac{\partial \mathcal{L}\left(\boldsymbol{\theta}^{(k+1)}(\lambda), \boldsymbol{X}^{(k+1)}\right)^T}{\partial \boldsymbol{\theta}} \widehat{\mathbf{G}}^{(k)} = -\mathbf{G}^{(k+1)T}\widehat{\mathbf{G}}^{(k)}\end{aligned}$$

Hence the gradient descent for the $\lambda$ parameter is:

$$\lambda^{(k+1)} = \lambda^{(k)} - \beta\left(-\mathbf{G}^{(k)T}\widehat{\mathbf{G}}^{(k-1)}\right) = \lambda^{(k)} + \beta\mathbf{G}^{(k)T}\widehat{\mathbf{G}}^{(k-1)}.$$

where $\beta$ is an update step for $\lambda$. The interpretation of the above is following, if $\mathbf{G}^{(k)T}\widehat{\mathbf{G}}^{(k-1)} > 0$ it means that gradients between two steps are pointing more or less in the same direction, in such case we can increase learning rate to speed up the convergence. However if $\mathbf{G}^{(k)T}\widehat{\mathbf{G}}^{(k-1)} < 0$ this may suggest that we have overshoot the minimum and we have to change direction. In such case the learning rate will be decreased to reduce the overshoot problem.

## 3.1 Potential problems with proposed method

- First of all, from the above one can see that $\lambda$ can be smaller than zero. Hence we propose following exponential update

$$\lambda^{(k+1)} = \lambda^{(k)}\left(1 + \beta\mathbf{G}^{(k)T}\widehat{\mathbf{G}}^{(k-1)}\right),$$

and we constrain the update to following condition $-1 < \beta \mathbf{G}^{(k)T} \widehat{\mathbf{G}}^{(k-1)} < 1$. Since $\beta$ is a hyperparameter, we can normalize $\mathbf{G}/\left\|\mathbf{G}\right\|_2$. Thus we have final update rule

$$\lambda^{(k+1)} = \lambda^{(k)} \left( 1 + \beta \frac{\mathbf{G}^{(k)T} \mathbf{G}^{(k-1)}}{\left\|\mathbf{G}^{(k)}\right\|_2 \left\|\mathbf{G}^{(k-1)}\right\|_2} \right),$$

and $|\beta| < 1$. Then we clip learning rate to certain range: $\lambda^{(k+1)} = \text{clip}\left(\lambda^{(\text{k}+1)}, \lambda_{\min}, \lambda_{\max}\right)$, this helps to avoid explosion of $\lambda$.

- Due to the stochastic nature of the SGD the correlation between two subsequent gradients may be disturbed by the randomness of the update process. We have observed that in real examples (CIFAR10 or MNIST) the value of the $\lambda_k$ decreased exponentially over time, which suggest that in most cases the correlation $\mathbf{G}_k^T \mathbf{G}_{k-1}$ is negative. I have tried to solve this problem by using Barzilai-Borwein approach proposed in Ref. [1], where the authors computes the exponential averages of gradients over certain number of samples. However this approach does not take into account that parameters of the model change during the optimization and the computed averages may not be meaningful. The Tensorflow implementation of the proposed Barzilai-Borwein Step Size method is available, however it may contain bugs.
- Possible solution for that is to better estimate gradients at each step. I have tried to increase the batch size but without success.

Because of the second problem the I have decided to use the manual learning rate scheduling. This approach combined with normalized gradients gave me the best results.

# 4 Final algorithm

In the final implementation each layer of the Neural Network has its own $\lambda_k$. See Algorithm 1.

# 5 Including momentum term

When adding momentum to the proposed normalized gradients one has to remember that the adaptive learning rate depends on the gradients correlation $\mathbf{G}^{(k)T} \mathbf{G}^{(k-1)}$. Hence, computing this term on averaged gradients is prohibited, since the correlation of running averages will be always positive and learning rate will grow exponentially. Momentum term is added in following way:

$$\mathbf{V}^{(k)} = m^{(k)} \mathbf{V}^{(k-1)} + \mathbf{G}^{(k)}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \lambda^{(k)} \frac{\mathbf{V}^{(k)}}{\left\|\mathbf{V}^{(k)}\right\|},$$

**Input:** A loss function $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{X})$ initial values of the model parameters $\boldsymbol{\theta}_l^{(0)}$ ($l$ denotes the index of the layer), initial value of learning rate for each layer in the model $\lambda_l^{(0)}$, $\mathbf{G}_l^{(-1)} = \mathbf{0}$, $N$ number of gradient descent steps and $\lambda_{\min} = 1e-6$, $\lambda_{\max} = 0.1$. $\|.\|$ is a selected normalizing function.
**for** $k = 0$ **to** $N$ **do**
Update parameters with

$$\boldsymbol{\theta}_l^{(k+1)} = \boldsymbol{\theta}_l^{(k)} - \lambda_l^{(k)} \frac{\mathbf{G}_l^{(k)}}{\left\|\mathbf{G}_l^{(k)}\right\|}.$$

Update learning rate with

$$\lambda_l^{(k+1)} = \text{clip}\left[\lambda_l^{(k)}\left(1 + \beta \frac{\mathbf{G}_l^{(k)T}\mathbf{G}_l^{(k-1)}}{\left\|\mathbf{G}_l^{(k)}\right\|_2 \left\|\mathbf{G}_l^{(k-1)}\right\|_2}\right), \lambda_{\min}, \lambda_{\max}\right].$$

Set $\mathbf{G}_l^{(k-1)} = \mathbf{G}_l^{(k)}$.
**end for**

**Algorithm 1:** Normalized gradient descent with adaptive learning rate

but the adaptive learning rate is using not current gradients $\mathbf{G}^{(k)}$ instead of $\mathbf{V}^{(k)}$. Note that momentum term can be learned in similar fashion as learning rate, hence we put explicit k dependency on $m^{(k)}$.

# 6 Things which were not working

- adaptive learning rate seemed to work only for non stochastic setup. When applied to the Neural network, the learning rate was decreasing monotonically resulting in slower training than with manual scheduling.
- Same conclusions for the momentum term. When using normalized gradient descent we have observed that adding momentum term does not bring any advantages when training in a stochastic way. The best result were obtained without momentum and adaptive learning rate.

# References

[1] Barzilai-Borwein Step Size for Stochastic Gradient Descent, Conghui Tan, Shiqian Ma, Yu-Hong Dai, Yuqiu Qian, (2016) https://arxiv.org/pdf/1605.04131.pdf